*Buletin Ukur*, Jld. 5, No. 2, ms. 89 -103, Mei, 1994


# A Program To Generalize Contour Lines

**Alias Abdul Rahman**
Jabatan Geoinformatik
Fakulti Ukur dan Harta Tanah
Universiti Teknologi Malaysia

**Abstract**

This paper describes a digitized points reduction algorithm, and reviews a number of existing algorithms. A complete computer program based on the developed algorithm is presented.

## 1.0 INTRODUCTION

Digitized contour lines is one of the common terrain data used to generate DTM. This digital contour data have to be filtered, compressed or reduced before it can be used to generate high fidelity DTM products. During contour line digitization, one may generate lot of redundant points which is unnecessary. These unwanted points have to be removed. It is desireable to have some of the points removed with the overall form of the contour lines maintained. Also, by having too large digitized data set, problems of storage space, and processing time are inevitable. It is important to realise that cartographic lines have been treated within a variety of geometric contexts, e.g. as a set of points, line, and even an areal(polygon) feature. Regardless what the context is, the whole idea is to maintain and preserve the accuracy and geographic or perceptual characteristics of the original line with the minimum number of points. If one can find a way to retain or to reject the points along the digitized lines, then the problem can be minimised.

## 2.0 SOME LINE GENERALIZATION ALGORITHMS

A number of the line generalization algorithms were reviewed e.g. by McMaster(1987), Jenks(1980), Zycor(1984), etc. There are quite a number of algorithms exist, namely, Douglas-Puecker, Parallel Tube, Perpendicular Distance, The N-th point algorithm, Distance Transverse, Moving Average, Angle Selection, etc. Some of the popular ones will be discussed here.

**The Douglas-Peucker algorithm** is based on the following principle: the method begins by defining the first point on the line as an anchor and the last point on the line as a floating point. A straight line segment is established from these two points. The other points which fall in between the two are examined right angle to that straight line. If the distance is less than the given tolerance, then the point is eliminated, other wise maintained. The point which produced the longest distance to the straight line now becomes the new floating point. Based on this new floating point and the previous anchor point, a new straight line is established. The same way of searching is applied on the current line until no more point has to be deleted(i.e. distance is greater than the specified tolerance). Then, move to the next line, see Figure 1.

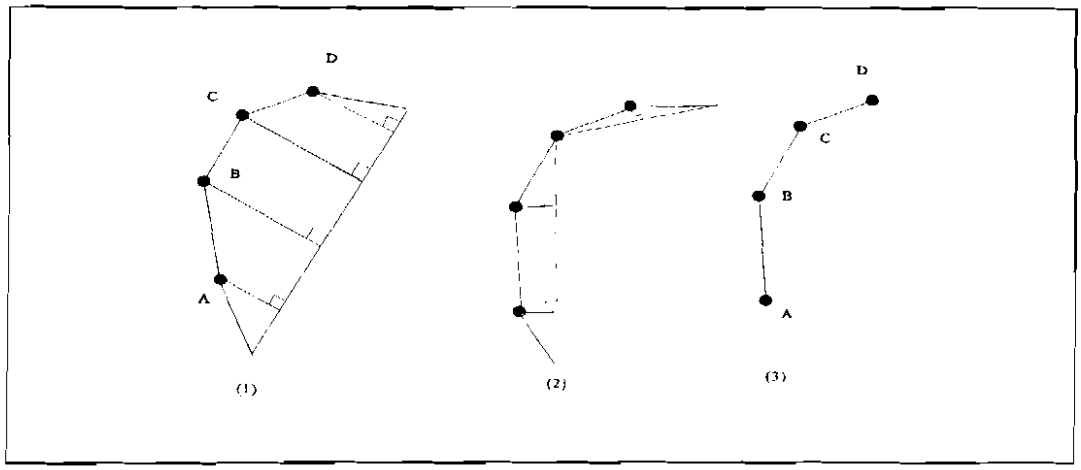


**Figure 1** The successive selection of points A, B, C, and D by the Douglas-Peucker algorithm

**Perpendicular Distance algorithm,** this method employs successive triad of points. The distance between the second point and the line formed by the first and third point is calculated. If the distance is smaller than the preset threshold, the middle point is eliminated, otherwise, it is retained. This procedure continues all along the line with every three consecutive points. This algorithm is simpler than the Douglas-Peucker one.

**The N-th. point algorithm,** is based on the following principle: that is every n-th point is retained to represent the original line. The level of generalization is proportional to the size of *n*. This method is simpler than the second one, and easy to implement in the computer.

**Angle Selection algorithm** is based on the following principle: that is, the angle between every two consecutive vectors of the line (i.e. three consecutive points) is computed. If the angle is bigger than the preset tolerance , the middle point is deleted, otherwise, the point is retained.

## 3.0 THE COMPUTER PROGRAM

The program works as follows: After interactive input of filenames and tolerances, the program reads coordinates of points from input file and writes out the coordinates of some of these points to the output file. The input file must be an ASCII list, where the data for each point are in one line and the first 3 numbers are the x, y, and z coordinateof the point separated by spaces or tabs. There may be text or more numbers behind the z-coordinate, but this is not use by the program. The output file is an ASCII list of 3 coordinates per line with the default format for real numbers(depend on the compiler used). There are two tolerances:

"line_tol" is the maximum distance by which the new polygon may deviate from the old polygon. This is also used as the minimum distance between output points.

"maxdist" is the maximum distance between output points.

In detail, the program:

(i) Reads "point 1"

(ii) It outputs "point 1"

(iii) It reads points until it finds one which is more than "line_tol" away from the first one. This point is "point 2".

(iv) It chooses one out of 8 directions as the "forward direction" ( the direction of the vector from "point 1" to "point 2", rounded to the nearest multiple of 45 degrees).

(v) It computes the maximum and the minimum azimuth ( with respect to the "forward direction") of vectors from "point 1", passing "point 2" at a distance of less than "line_tol". (Actually by simplifications in the computation the limits are slightly narrower than stated here!).

(vi) If such limits already exist, it chooses the smaller one from the old and the new maximum and the larger one from the old and the new minimum. (Narrowest limit is chosen).

(vii) It reads points until it finds one which is more than "line_tol" away from "point 2". This point is "point 3".

(viii) If the z-coordinate of "point 3" is different from the z-coordinate of "point 2", "point 2" is output, "point 3" becomes "point 1" and for the last line it outputs on the terminal the number of points read, the number of points written and the percentage of points written. Then it resumes from step (ii). (Start of a new contour line, because z is changed.)

(ix) If "point 3" is "inside", then "point 3" becomes "point 2" and the previous "point 2" is thus forgotten. If "point 3" is not "inside", then "point 2" is output and becomes "point 1", "point 3" becomes "point 2", and the program resumes from step (iv).

"point 3" is "inside" if it fulfills the following 3 conditions:

- the azimuth from "point 2" to "point 3" may not deviate more than 90 degrees from the "forward direction".
- the distance from "point 1" to "point 3" may not be more than "maxdist".
- the azimuth from "point 1" to "point 3" is inside the limits established in steps (v) and (vi).

The program continues until the end of the input file. It then outputs the total number of point read, the total number of points written and the percentage of points written.

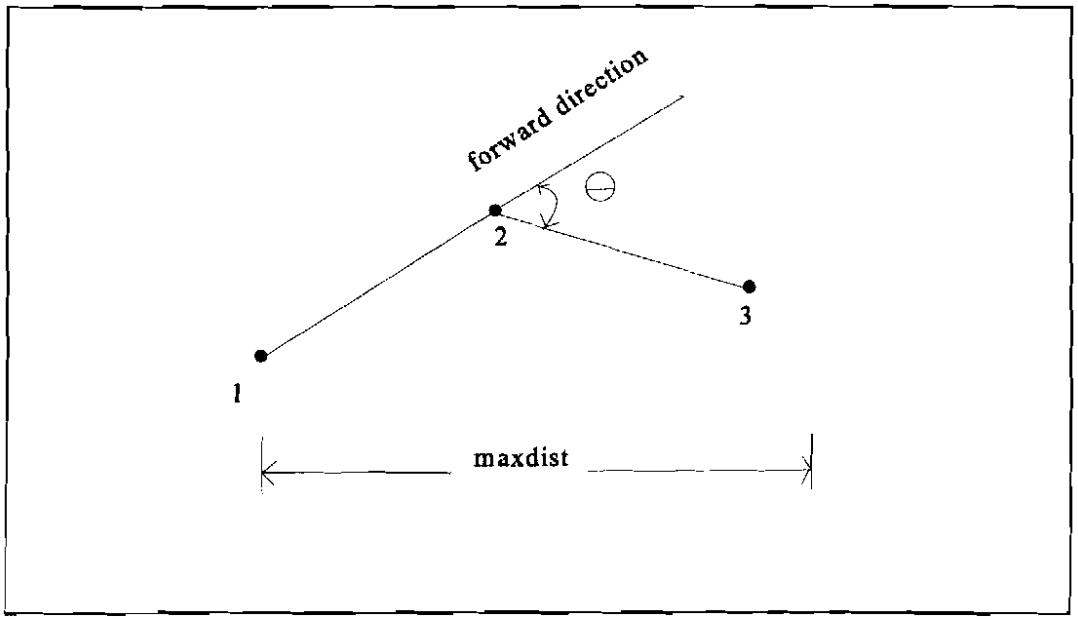


**Figure 2** The point reduction algorithm

The general idea of this program is the following: The input file is a coordinate list of the nodes of line-polygons. The program copies these points to the output file omitting points such, that no one of the omitted points is(in planimetry) further than the tolerance away from the line-polygon defined by the remaining points. Many DTM conversion programs fail when contour data are too sparse, therefore points will not be omitted if the distance otherwise becomes larger than a user-defined limit.

The algorithm can be presented as in the following Pascal-like statements:

```
get_point(1);
        while point_found do
                put_out(1);
                get_point(2);
                if point_found then
                        set_direction;
                        while point_found not last_point do
                                set_limits
                                get_point(3);
                                if point_found then
                                        if point_inside then
                                        swap(2,3)
                                else
                                        put_out(2);
                                        shift_up; { shift to the next point }
                                        set-direction
end;
```

## 4.0 TEST AREA

The test area is around Kluang district, Johor, covers an area of approximately 20 km by 20 km (i.e. a sheet of topographic map at the scale of 1:25,000). The relief of the area is mainly dominated by gentle sloping terrain.

## 5.0 TESTING PROCEDURE

The algorithm was tested using the following procedure:

- Contour data (in Arc/Info LIN format) is converted to an ASCII file of x, y, z coordinates.
- The file of x, y, z coordinates is used as an input file to the program called CONTRED.EXE (i.e. the generalization program), and produces the same file format for the output.
- The output file then reconverted to the Arc/Info format for display and visualisation purposes.

With the tolerance of 0.2 cm (digitizer coordinate), 69.2% of the original data set were reduced (i.e. 5903 points were removed from the total digitized points of 8525). If we compare the two plots visually in Figure 4, then we can say that though nearly 70% of the points were eliminated, but still the overall appearence of the contours were maintained. Thus, meets one of the generalization rules.
The testing flow chart is given in Figure 4.

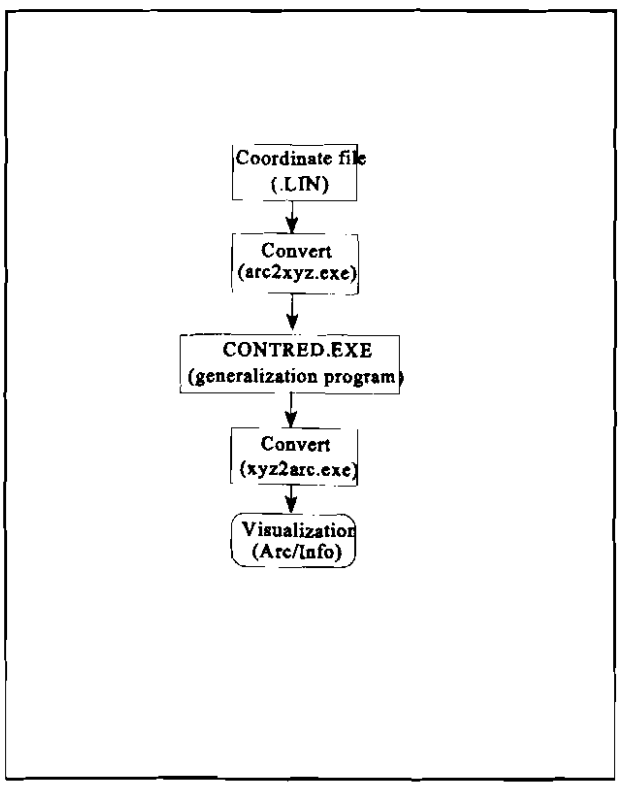


**Figure 3** The testing flow chart.

## 5.0 DISCUSSION

From the above experimental test, it has been shown that the program which is based on the developed algorithm produces acceptable results, see Figure 4. The program can be improved e.g by changing the way the input data file is read and written to the output file as line strings(i.e. with a code for each string) instead of typical list of x,y,z coordinates.

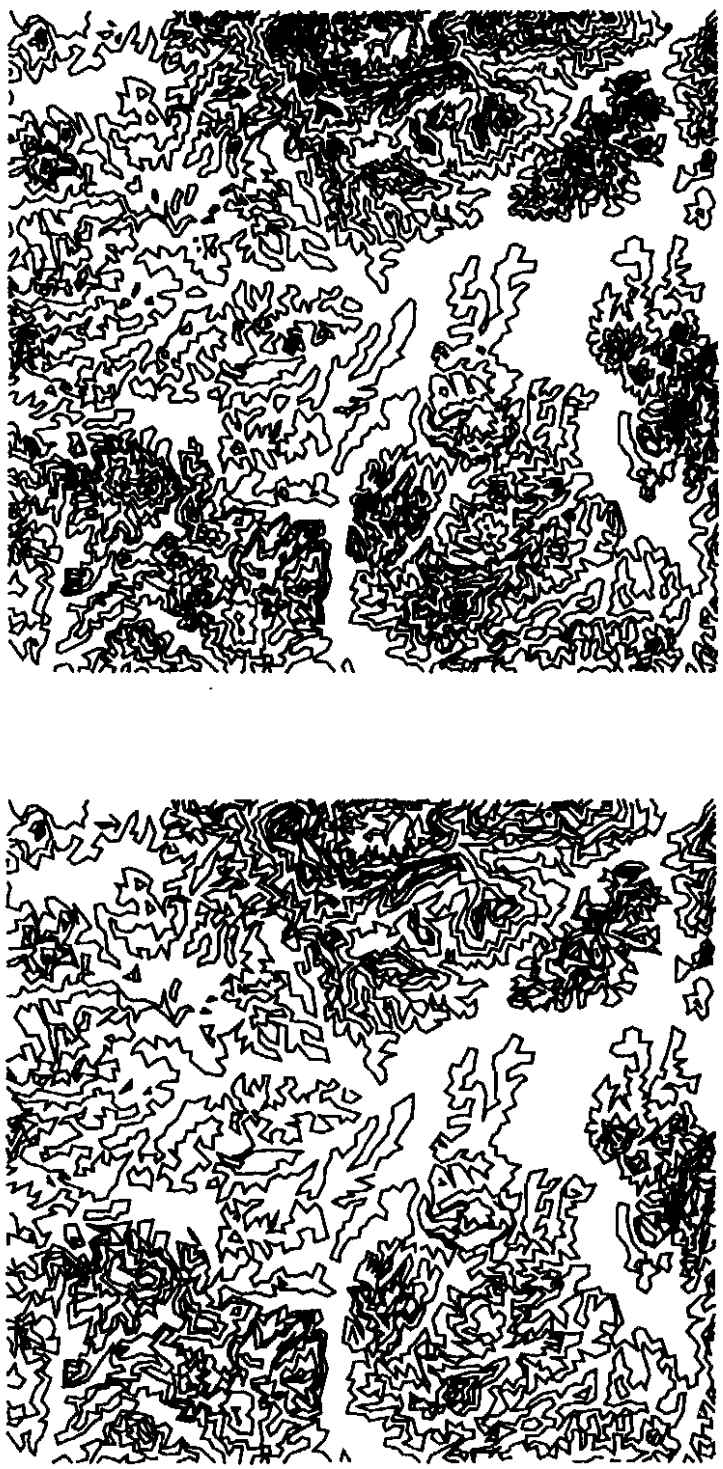

**Figure 4** Top: the original contour data set(with 8525 data points), Bottom: contours after the execution of the CONTRED.EXE program(with 2622 remained data points).

## References

JENKS, (1980), Thoughts on line generalisation, *Proceedings of the Int. Symposium on Cartography and Computing, AUTOCARTO IV*, ACSM & ASP, pp. 209-220

McMASTER, (1987), Automated Line Generalization, *Cartographica*, Vol. 24, No. 2, pp. 74-111.

ZYCOR, (1984), Manual and automated feature displacement, *Report of the U.S. Army Engineer Topographic Laboratories*, Fort Belvoir, Virginia, 204 p.

**Acknowledgement**
The author would like to thank Dipl. Ing. Karl Grabmaier from Photogrammetry Division, Department of Geoinformatics, ITC, Enschede, The Netherlands for his initial idea and invaluable discussions. The author also appreciate the review effort put by Dr. Mohammed Said Mat Lela, colleague and senior lecturer from the Department of Geoinformatics, Faculty of Surveying and Real Estate, Universiti Teknologi Malaysia.

## About the author

Alias Abdul Rahman is a lecturer in the Department of Geoinformatics, Faculty of Surveying and Real Estate, UTM. He received his Post Graduate Diploma and M.Sc. in Integrated Map and Geoinformation Production from ITC, Enschede, The Netherlands in 1990 and 1992 respectively. His research interests focus on DTM data processing algorithms development, and DTM/GIS integration.

## Appendix A

This is the complete listing of the program to generalize contour lines. It is written in Turbo Pascal version 5.5

```
program ContourDataReduction(input, output);
 uses
   crt ;

 const
   title: array[1..8] of string=
        ('+-------------------------------------------------------------+',
         '|                                                             |',
         '|  *** Program To Generalize Contour Lines ***    |',
         '|                                                           |',
         '|    (c)Copyright 1991 by Alias Abdul Rahman       |',
         '|                                                           |',
         '|               All rights reserved                      |',
         '+-------------------------------------------------------------+');

type
   pointer_range          = 1..3;
   pointer_array_of_real  = array[pointer_range] of real;
   pointer_array_of_pointer = array[pointer_range] of pointer_range;

var
   direction                    : integer;
   incount, outcount,
   in_total, out_total          : integer;
   dx, dy, limit1, limit2       : real;
   line_tol, maxdist            : real;
   limit, last_pt, point_found  : boolean;
   x, y, z                      : pointer_array_of_real;
   pointer                      : pointer_array_of_pointer;
   infile, outfile              : text;
   infilename, outfilename      : string;

procedure pause;
   begin
     writeln;
     write('Press <ENTER> to continue');
     readln;
   end; { pause }

procedure copyright;
  var
    i : integer;
    begin
      clrscr;
      gotoXY(1,10);
      for i := 1 to 8 do
      writeln(Title[i]:62);
      pause;
      clrscr;
    end; { copyright }
```

```
procedure openInputFile( var Infilename : string);
  begin
     write('Enter input filename >');
     readln(Infilename);
     assign(infile, Infilename);
     reset(infile);
  end;{ openInputFile }

procedure openOutputFile( var outFilename : string);
  begin
     write('Enter output filename >');
     readln(outFilename);
     assign(outfile, outFilename);
     rewrite(outfile);
  end; { openOutputFile }

function forwad( from, towards : integer) : boolean;
  begin
     dx := x[pointer[towards]] - x[pointer[from]];
     dy := y[pointer[towards]] - y[pointer[from]];
     case direction of
        1 : forwad := dx > 0;
        2 : forwad := dx > -dy;
        3 : forwad := dy > 0;
        4 : forwad := dx < dy;
        5 : forwad := dx < 0;
        6 : forwad := dx < -dy;
        7 : forwad := dy < 0;
        8 : forwad := dx > dy;
     end;
  end; { forwad }


procedure prepare;
  var
    maxlength, i  : integer;

  begin
     for i := 1 to 3 do
       pointer[i] := i;
       openInputFile(infilename);
       write('Enter the tolerance for omitting points :');
       readln(line_tol);
       writeln;
       write('Enter the maximum distance between output points :');
       readln(maxdist);
       writeln;
       openOutputFile(outfilename);
       incount := 0;
       outcount := 0;
       in_total := 0;
       out_total := 0;
       point_found := true;
       last_pt := false;
  end; { prepare }
```

```
procedure shift_up;
  var
   free  : integer;

   begin
     free := pointer[1];
     pointer[1] := pointer[2];
     pointer[2] := pointer[3];
     pointer[3] := free;
   end; { shift_up }

procedure swap( pt1, pt2 : integer);
  var
   help  : integer;

  begin
    help := pointer[pt1];
    pointer[pt1] := pointer[pt2];
    pointer[pt2] := help;
  end; { swap }


procedure put_out( pt : integer);
  begin
    writeln(outfile, x[pointer[pt]]:10:3, y[pointer[pt]]:10:3, z[pointer[pt]]:10:3);
    outcount := succ(outcount);
  end; { put_out }

function distance( from, onto : integer ) : real;
  begin
    dx := x[pointer[onto]] - x[pointer[from]];
    dy := y[pointer[onto]] - y[pointer[from]];
    distance := sqrt(dx*dx + dy*dy);
  end; { distance }

procedure get_point (pt  : integer);
  var
   this_point : integer;

  procedure read_point;
    begin
      point_found := not eof(infile);
      if point_found then
        begin
          readln(infile, x[this_point], y[this_point], z[this_point]);
          incount := succ(incount);
        end;
    end; { read_point }

  begin { get_point }
    this_point := pointer[pt];
    if (pt = 1) then
      read_point
    else
      repeat
```

```
        read_point;
      until (distance(pt, pt-1) > line_tol) or not point_found;
  end; { get_point }


procedure set_direction;
  const
    tan_pi_8 = 0.41421;

  var
    dylim, dxlim : real;

  begin { set_direction }
    dx := x[pointer[2]] - x[pointer[1]];
    dy := y[pointer[2]] - y[pointer[1]];
    dylim := tan_pi_8 * dx;
    dxlim := tan_pi_8 * dy;
    if (dy > -dylim) then
      if (dx > dxlim) then
        if (dy < dylim) then
          direction := 1
        else
          direction := 2
        else
          if (dx > -dxlim) then
            direction := 3
          else
            direction := 4
          else
            if (dx < dxlim) then
              if (dy > dylim) then
                direction := 5
              else
                direction := 6
              else
                if (dx < -dxlim) then
                  direction := 7
                else
                  direction := 8;
      limit := false;
  end; { set_direction }
```

```
function tangent : real;
  begin
    case direction of
      1, 5 : tangent := dy/dx;
      2, 6 : tangent := (dy-dx)/(dx+dx);
      3, 7 : tangent := dx/dy;
      4, 8 : tangent := (dx+dy)/(dy-dx);
    end;
  end; { tangent }
    { tangent of the angle with the "forward direction" }


procedure set_limits;
  var
   lim1, lim2, k1, k2, kk : real;

  begin
    k1 := tangent;
    k2 := line_tol / distance(1, 2);
    kk := k1 * k2;
    lim1 := (k1+k2) / (1.0-kk);
    lim2 := (k1-k2) / (1.0+kk);
    if limit then
      begin
        if lim1 < limit1 then
          limit1 := lim1;
        if lim2 > limit2 then
          limit2 := lim2;
      end
      else
        begin
          limit1 := lim1;
          limit2 := lim2;
          limit := true;
        end;
  end; { set_limits }
```

```
function inside : boolean;
  var
    ok : boolean;
    k1 : real;

  begin
     last_pt := (z[pointer[3]] <> z[pointer[2]]);
     if last_pt then
       inside := false
     else
       begin
          ok := forwad(2, 3) and (distance(1, 3) < maxdist);
          if ok then
            begin
               k1 := tangent;
               ok := (k1 < limit1) and (k1 > limit2);
            end;
          inside := ok;
       end;
  end; { inside }
     { inside, next point has the same Z value as the previous one
       and is inside the permitted area, which is :
       - forward from point 2
       - not too far from point 1
       - inside the limitting rays defined by limit1 and limit2
     }

procedure endline;
  begin
     put_out(2);
     writeln(outcount:4, ' from ', incount:4, ' (',
           outcount * 100.0 / incount:3:1, '%)' );
     in_total := in_total + incount;
     out_total := out_total + outcount;
     outcount := 0;
     incount := 0;
     swap(1, 3);
     last_pt := false;
  end; { endline }
```

```
begin { main program }
  copyright;
  prepare;
  get_point(1);
  while point_found do
    begin
      put_out(1);
      get_point(2);
      if point_found then
        begin
          set_direction;
          while point_found and not last_pt do
            begin
              set_limits;
              get_point(3);
              if point_found then
                begin
                  if inside then
                    swap(2, 3)
                  else
                    if not last_pt then
                      begin
                        put_out(2);
                        shift_up;
                        set_direction;
                      end;
                end;
            end;
        end;
        endline;
    end;
    writeln(' in total : ',out_total:4, ' from ', in_total:4,
        ' (', out_total * 100.0 / in_total:3:1, '%)' );
    writeln;
    writeln('Press <ENTER> to continue');
    readln;
    close(infile);
    close(outfile);
end. { main program }
```